Session 138

CoreFoundation

Chris Kane
Cocoa Frameworks Group

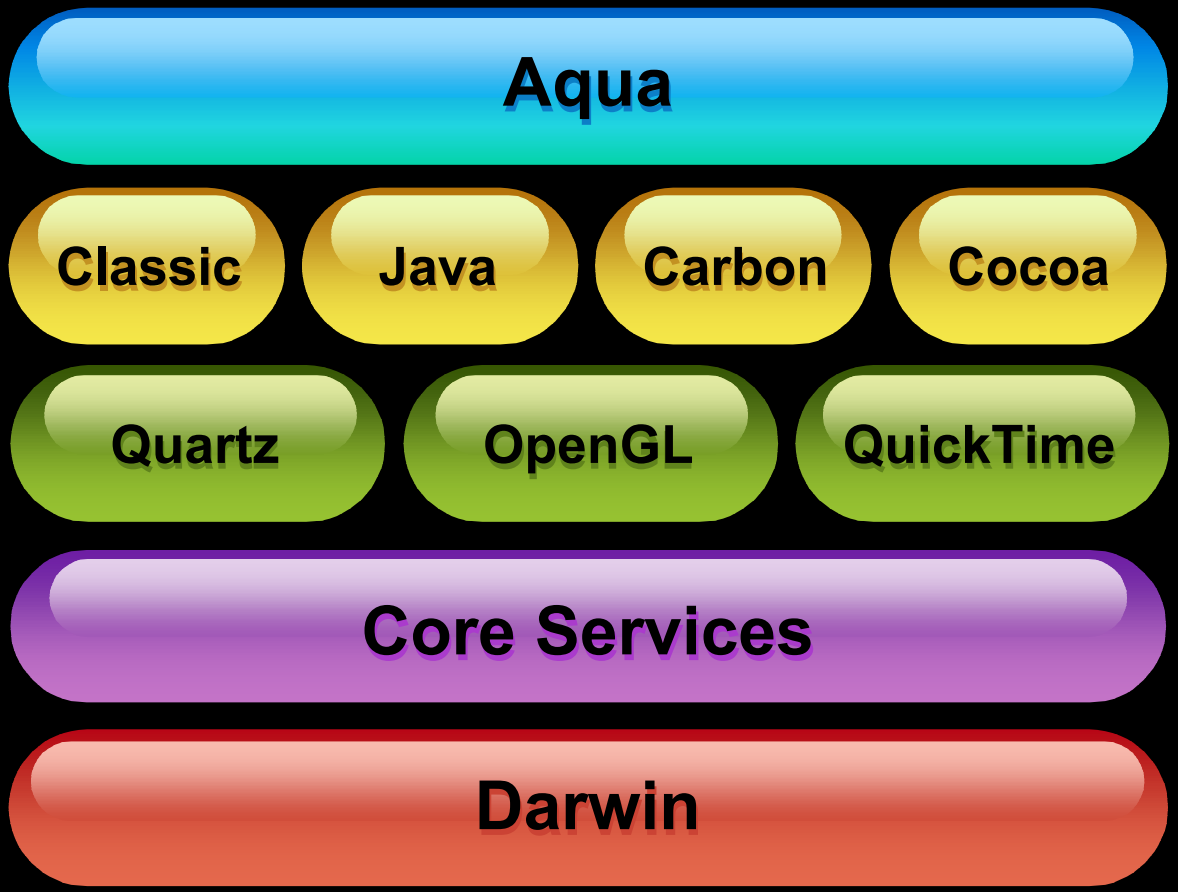
Introduction

- What is CoreFoundation?
- Basic data types
- Concepts and models
- Explore some CoreFoundation services

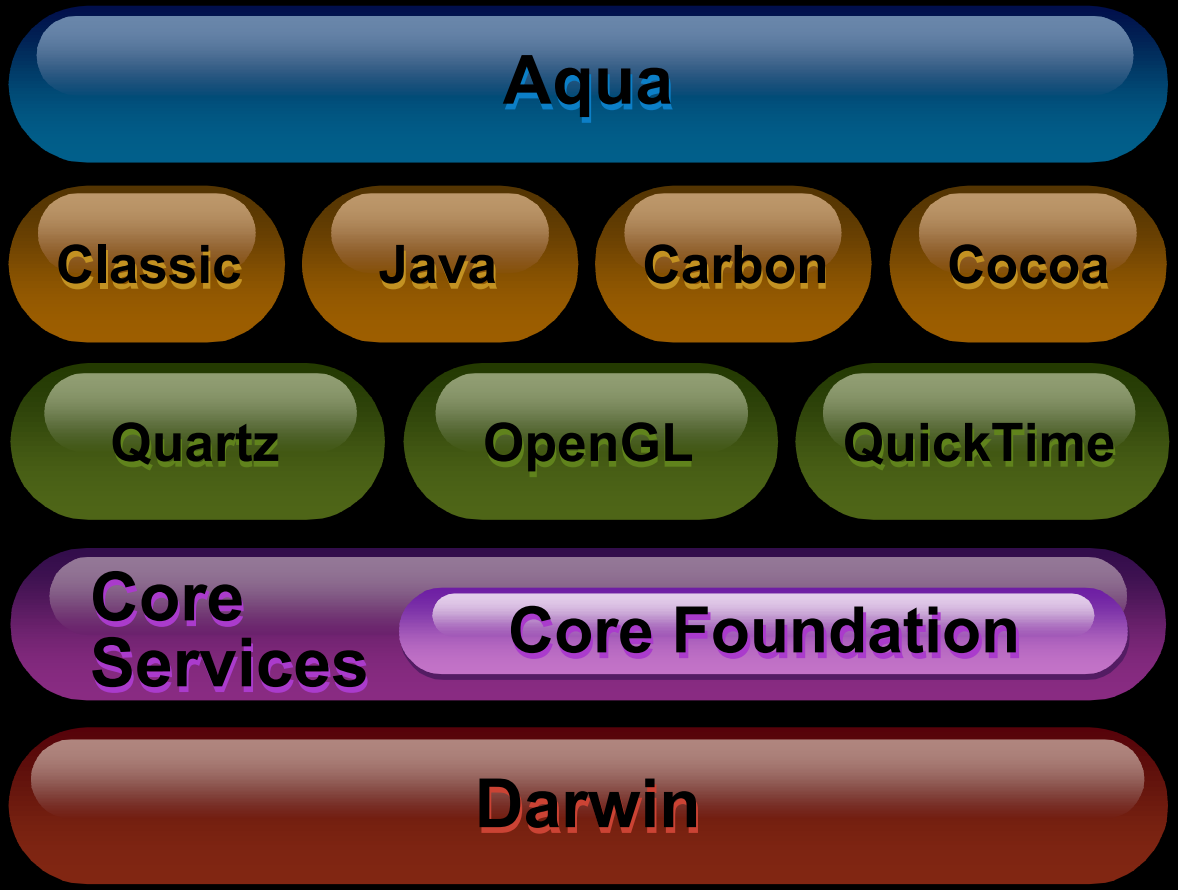
Core Foundation

- Substrate framework for Mac OS X
  - Used by both Carbon and Cocoa stacks
  - CarbonLib for Mac OS 8/9 contains a subset
  - Darwin contains a subset
- C APIs
- Also called “CF”

Structure of Mac OS X



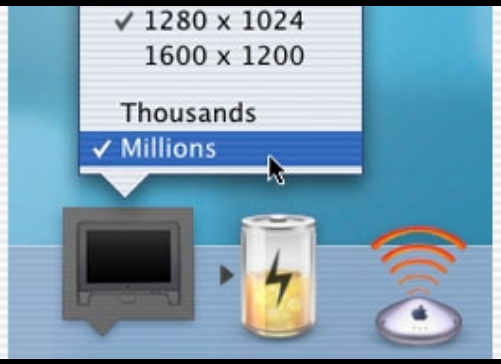
Structure of Mac OS X



CoreFoundation Contains…

- Basic data types
  - Strings, collections, dates
- App services
  - App packaging, localization
  - User preferences
- … And the rest
- No graphics

Basic Data Types



CFString

- Array of Unicode characters
- Elevates strings to a new level of abstraction
  - Make internationalization easy
  - 8-bit encoding savvy
  - The way to exchange strings in APIs
- New APIs using strings will use CFStrings

CFString Functionality

- Many ways to create CFStrings
- String formatting similar to printf()
- Encoding conversions
- Appending, concatenation
- Comparison, find

Other Primitive Types

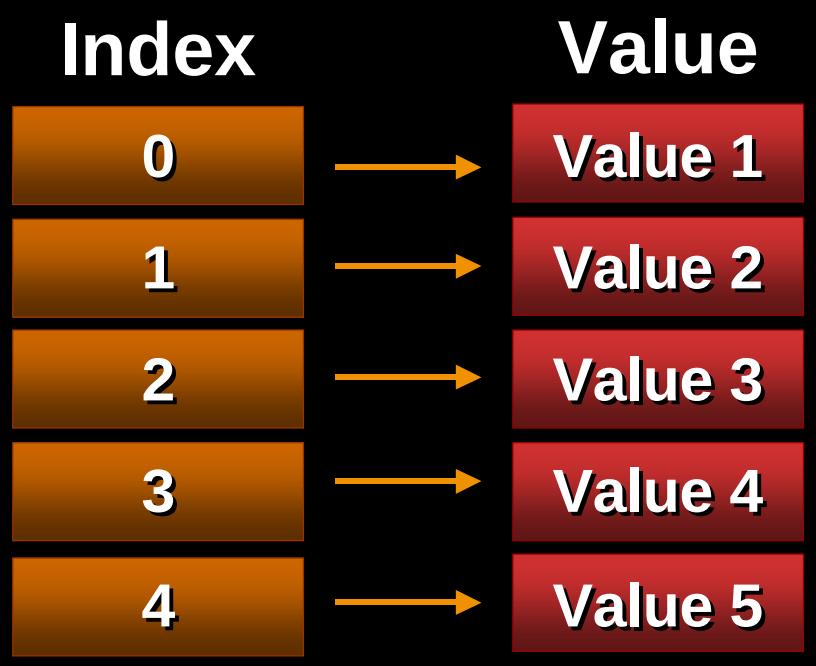
- CFData: byte array with length
- CFNumber: wrapper for C arithmetic types
- CFBoolean: true & false
- CFDate: points in time, date computations
- CFURL: represent URLs and file names

Collections

- Containers for pointer-sized values
  - Primarily for CF types
- Configured via callback functions
  - Callbacks provided at creation
  - Determine how values are compared, added, and removed

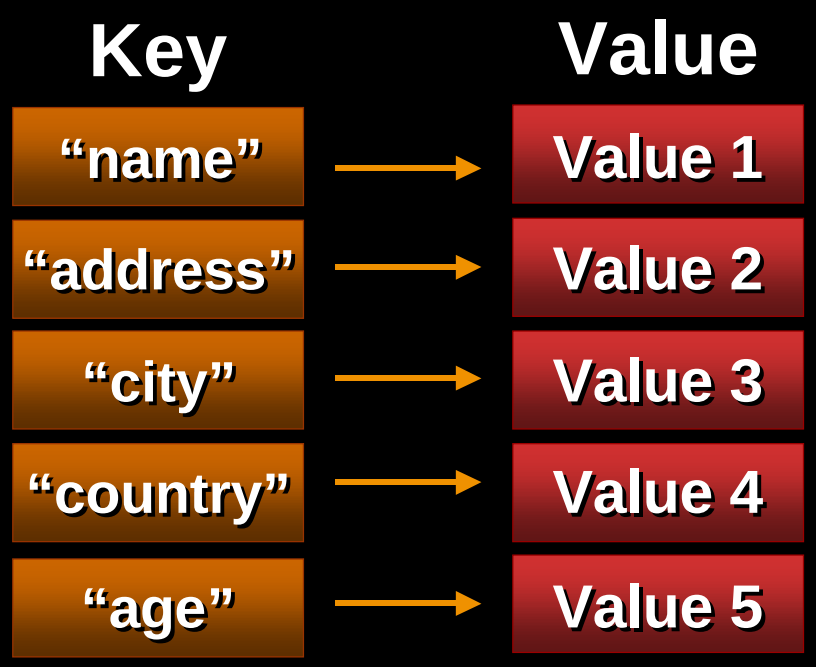
CFArray

- Ordered list of values
- Values looked up via index
- Indices are zero-based
- Values can be nearly any type of quantity



CFDictionary

- Set of key/value pairs
- Values looked up via keys
- Based on hashing of the keys
- Keys and values can be nearly any type of quantity



Property Lists

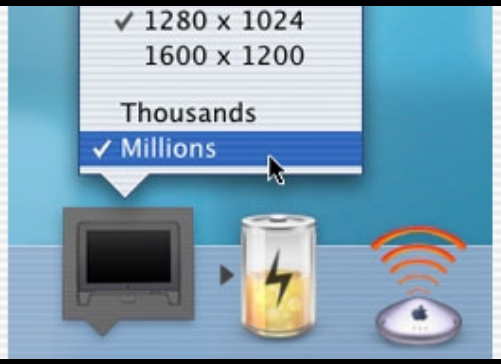
- Any object of these types, or collection of these, only:
  - CFString, CFData, CFNumber, CFBoolean, CFDate, CFArray, or CFDictionary
  - Dictionary keys must be CFStrings
- Have a flattened XML representation
- Used for configuration info, data files, user preferences

Property Lists

<CFDictionary>

“Printer Name”	“foobar”
“Print Date”	2001-05-25T16:00:00Z
“Show Panel”	false
“Paper Name”	“A4”
“Paper Width”	21.0
“Paper Height”	29.7
“Orientation”	“Landscape”
“Copies”	3

CoreFoundation Concepts



Concepts and Conventions

- General philosophy
- API conventions
- Polymorphism
- Memory management
- Toll-free bridging

General Philosophy

- High performance, yet configurable
- API consistency with conventions
- Designed around object-oriented paradigms
  - Opaque data types
  - Groups of functions to operate on them
  - Small set of polymorphic functions

General Philosophy

- No safety net
  - Little checking of programming errors
  - Debug library to catch common errors

API Conventions

- Each type is opaque and acts as a pseudo-class
  - CF<class>Ref

    CFStringRef
- Related functions act as methods
  - CF<class><action>

    CFStringAppendFormat()
- First argument is the type instance (the receiver)
  - Exception: **Create** functions

Polymorphic Functions

- A small number of functions can be used with any CF type
- Equality/hashing
  - CFEqual(), CFHash()
- Introspection
  - CFGetTypeID(), CFCopyDescription()
- Memory management
  - CFRetain(), CFRelease(), CFGetRetainCount(), CFGetAllocator()

Memory Management

- All CF types are reference counted
- CFRetain() to take a reference; CFRelease() to release it
- Any code which retains an object is part-owner of that object
- Fundamental question: who “owns” an object
  - Example: if a function returns a CF type, who has the reference?

Memory Management

- Functions with the verb **Get** do not give the caller a reference
  - Retain if you wish to keep the object
- Functions with the verb **Copy** do give a reference
  - Release when you are finished with it
- Functions with the verb **Create** return new instances
  - Release when you are finished with it

Memory Management

- **Copy** functions might not perform any memory copies
  - But will always return a reference
- **Create** functions might not perform any allocation
  - But will always return a reference
- CoreFoundation is just being more efficient; this shouldn't matter to the caller

Toll-free Bridging

- Enables low-level objects to be passed between the stacks
- Some CF types are the “same” as their Cocoa counterparts
  - CFString, CFArray, CFDictionary, CFData, . . .
- Allows passing these objects between Cocoa and Carbon APIs without any conversion cost

```
CFStringRef cfstr = (CFStringRef)nsstring;
NSString *nsstring = (NSString *)cfstr;
```

Application Services

Doug Davidson
Cocoa Frameworks Group

Basic App Services

- CFBundle
  - App packaging, localization, code loading
- CFPlugIn
  - Plugin management
- CFPreferences
  - User preferences

CFBundle

- Collects together:
  - Executables
  - Localized and non-localized resources
  - Property list for metadata
  - Arbitrary files and folders
- Can have multiple executables
  - Including separate versions for Mac OS 9 and Mac OS X

CFBundle

- Look up localized versions of strings and other resources
  - Uses user's language preferences
- Load code from a bundle at runtime
  - On Mac OS X, can be Mach-O or CFM
- Look up functions by name
  - Useful for cross-ABI function pointers

CFPlugIn

- Specialized type of CFBundle
- Manages interfaces for plugins
- Clients obtain interfaces by UUIDs
- Interfaces are COM-compatible tables of function pointers
- Best learned by example, in the CoreFoundation documentation and CarbonLib SDK

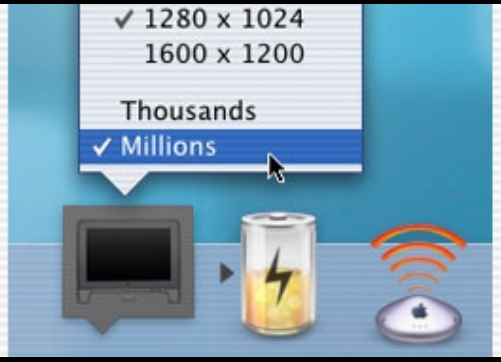
CFPreferences

- For easily storing and retrieving user preferences
- Preferences usually per user and per app
- Each preference has a CFString name
- Value can be any property list value
  - CFString, CFNumber, CFDictionary, . . .
- CFPreferences handles everything else
  - Takes care of data format and storage
  - Currently uses XML for most backing stores

CFPreferences Notes

- Not for 100s of K of data
  - Especially large binary data
  - Read-only data private to the application should go inside the app bundle
- Apps should use preferences robustly
  - Your app should not crash because of a bad or missing value

Advanced Application Services



Advanced App Services

- CFRunLoop: event handling
- CFRunLoopTimer: delayed events
- CFMessagePort: point-to-point IPC
- CFNotificationCenter: broadcast IPC

CFRunLoop

- Enables advanced services
- Event manager and dispatcher
  - Where “event” is intended to be understood broadly
- Maintains sets of event sources
- Callbacks to registered handlers when events occur

Run Loops

- Each thread has exactly one run loop
- CFRunLoop is CoreFoundation interface to it
- Apps will usually use Carbon Events or AppKit interfaces
- Use the highest-level API that meets your needs
- CFRunLoop for CoreFoundation-based sources

Run Loop Modes

- Run loop event-source sets are called “modes”
- Modes are used to restrict which event sources are monitored
  - Example: **NSModalPanelRunLoopMode**
- Normally you will use and run the run loop in the default mode
- Can also use “common” modes

  kCFRunLoopCommonModes

CFRunLoop Sources

- Some provided source types
  - CFRunLoopTimer
  - CFMachPort
  - CFMessagePort
  - CFSocket
- You can create your own, but it can be complex

CFRunLoopTimers

- One-shot or repeating
- Callback function provided at create time
- Add it to a run loop with modes
- When date arrives, it will fire
  - But only when the run loop is running
- If one-shot, automatically invalidated

CFMessagePort

- Low-overhead, high-performance IPC
- Between threads or processes on the same machine
- Messages can be sent
  - One-way and asynchronous
  - With reply and synchronous

CFMessagePort

- Local port is your end

  CFMessagePortCreateLocal()

- Local ports may be named and advertised with a CFString
- Remote port is the other end, looked up by name

  CFMessagePortCreateRemote()

Sending Messages

- Messages are a CFData and integer message ID
- Optional replies are CFData
- Receiver defines and publishes the protocol for messages
  - The format of the data
  - What the message IDs mean
  - If there will be a reply

Messaging Process

- Sender sends a message

  CFMessagePortSendRequest()
- Receiver's callback is called
- Receiver returns optional reply
- Sender gets reply

  CFMessagePortSendRequest() returns
- Can optionally receive other messages or events while waiting

CFNotificationCenter

- Robust broadcast IPC mechanism
- Between processes on the same machine
- Broadcast events to interested apps
  - Called “posting a notification”
- Sender need not know receivers

Notification Posting

- Notifications have a name, an optional string (“object”), and an optional property list dictionary (“user info”)
- Sender defines and publishes the protocol for its notifications
  - Meaning of named notifications
  - Format of user info
- Sender posts notification and continues
- Sender may choose immediate delivery

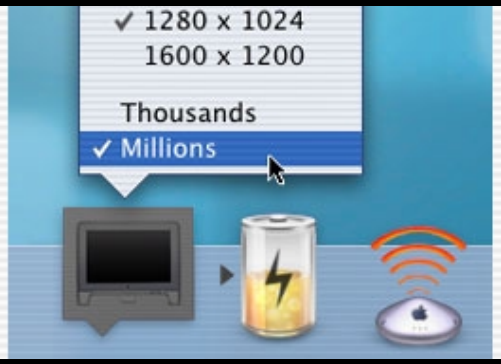
Notification Listening

- Interested apps register for notifications
  - Based on name and/or string
- Options determine whether the notification is received immediately or delayed
- Optional coalescing of queued notifications
- Run loop must be running for notifications to be received

CFNotificationCenter Notes

- Not a high-volume mechanism
  - Immediate notification delivery particularly stressful on the system
- Point-to-point IPC is better if you know the receivers
- Especially useful in combination with other channels
  - Example: preferences changed notification

Documentation



Documentation

- On-line documentation
  http://developer.apple.com/techpubs/corefoundation
  - **/Developer/Documentation/CoreFoundation**
- Example code
  /Developer/Examples/CoreFoundation
  - CarbonLib SDK Examples

Release Notes

- In **/Developer/Documentation/ReleaseNotes**
  - CoreFoundation.html
  - CFBundleAndCFPlugIn.html
  - InfoPlist.html
  - Localization.html

Wrap Up

Mark Turner
Carbon Technology Manager
Apple Worldwide Developer Relations

Who to Contact

Mark Turner
Carbon Technology Manager
Apple Worldwide Developer Relations
mark@apple.com

Feedback
For comments and requests, not for questions
corefoundation-feedback@group.apple.com

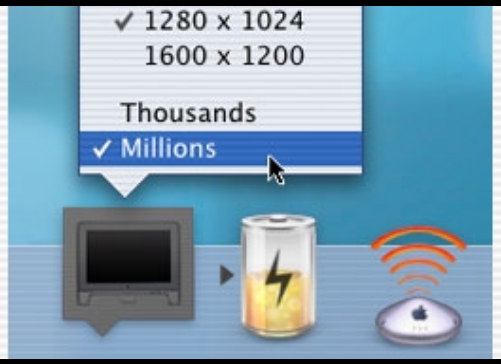
Public Mailing Lists
Two possibilities, for questions
cocoa-dev@lists.apple.com
carbon-developer@lists.apple.com

http://developer.apple.com/events/urls.html

Roadmap

102 Cocoa Overview: Introduction to the Cocoa Frameworks	Hall 2 **Mon., 5:00pm**
106 Carbon Event Manager: Carbon interface to CFRunLoop	Hall 2 **Tue., 5:00pm**
123 Advanced Cocoa Topics: Run Loop and IPC in Cocoa	Hall 2 **Thurs., 5:00pm**
FF112 Cocoa Feedback Forum: Feedback for Cocoa, will take CF feedback also	Room A2 **Fri., 5:00pm**

Q&A



Mark Turner
Carbon Technology Manager
mark@apple.com

http://developer.apple.com/events/urls.html

WWDC
Worldwide Developers Conference 2001